PDE-BASED GRADIENT LIMITING FOR MESH SIZE FUNCTIONS

Per-Olof Persson

Dept. of Mathematics, Massachusetts Institute of Technology, persson@math.mit.edu

ABSTRACT

We propose a new method for limiting the gradients in a mesh size function by solving a non-linear partial differential equation on the background mesh. Our gradient limiting Hamilton-Jacobi equation simplifies the generation of mesh size functions significantly, by decoupling size constraints at specific locations from the mesh grading requirements. We derive an analytical solution for convex domains which shows the results are optimal, and we describe how to implement efficient solvers on various types of meshes. We demonstrate our size functions with a proposed new mesh generation algorithm, using examples with curvature, feature size, and numerical adaptation.

Keywords: mesh generation, size function, background grids, Hamilton-Jacobi, gradation control

1. INTRODUCTION

In many mesh generation algorithms it is advantageous if an appropriate mesh size function h(x) is known prior to computing the mesh. This includes the advancing front method [1], the paving method for quadrilateral meshes [2], and smoothing-based mesh generators such as the one we proposed in [3]. The popular Delaunay refinement algorithm [4], [5] typically does not need an explicit size function since good element sizing is implied from the quality bound, although the mesh size function is a crucial underlying aspect of the algorithm that is sometimes obscured in the analysis.

Several techniques have been proposed for automatic generation of mesh size functions, see [6], [7], [8]. A common solution is to represent the size function in a discretized form on a background grid [1], and the actual values of $h(\boldsymbol{x})$ are obtained by interpolation. The function should take into account geometrical features, such as curvature and boundary proximity, as well as user-defined size constraints and size estimates from numerical adaptive solvers.

An important requirement on the size function is that the ratio of neighboring element sizes in the generated mesh is less than a given value G. This corresponds to a limit on the gradient $|\nabla h(x)| \leq g$ with g = G - 1 (or possibly $g = \log G$, depending on the interpretation). In some simple cases, this can be built into the size function explicitly. For example, a "point-source" size constraint $h(y) = h_0$ in a convex domain can be extended as $h(x) = h_0 + g|x - y|$, and similarly for other shapes such as edges. For more complex boundary curves, local feature sizes, user constraints, etc, such an explicit formulation is difficult to create and expensive to evaluate. It is also harder to extend this method to non-convex domains (such as the example in Figure 3), or to non-constant g (Section 5.2).

One way to limit the gradients of a discretized size function is to iterate over the edges of the background mesh and update the size function locally for neighboring nodes [9]. When the iterations converge, the solution satisfies $|\nabla h(x)| \leq g$ only approximately, in a way that depends on the mesh (see Section 4.3). Another method is to build a balanced octree, and let the size function be related to the size of the octree cells [10]. This data structure is used in the quadtree meshing algorithm [11], and the balancing guarantees a limited variation in element sizes, by a maximum factor of two between neighboring cells. However, when used as a size function for other meshing algorithms it

provides an approximate discrete solution to the original problem, and it is hard to generalize the method to arbitrary gradients g or different background meshes.

We present a new technique to handle the gradient limiting problem, by a continuous formulation of the process as a Hamilton-Jacobi equation. Since the mesh size function is defined as a continuous function of x, it is natural to formulate the gradient limiting as a PDE with solution h(x) independently of the actual background mesh. We can see many benefits in doing this:

- The analytical solution is exactly the optimal gradient limited size function h(x) that we want, as shown by Theorem 2.1. The only errors come from the numerical discretization, which can be controlled and reduced using known solution techniques for hyperbolic PDEs.
- By relying on existing well-developed Hamilton-Jacobi solvers we can generalize the algorithm in a straightforward way to
 - Cartesian grids, octree grids, or fully unstructured meshes
 - Higher order methods
 - Space and solution dependent q
 - Regions embedded in higher-dimensional spaces, for example surface meshes in 3-D.
- We can compute the solution in $\mathcal{O}(n \log n)$ time using a modified fast marching method.

In Section 2 we present the gradient limiting equation and some of its properties. Next we discuss various implementations, in particular a fast algorithm with nearly linear computational complexity. In Section 4 we show several examples of meshes generated using our size functions. We show how to use gradient limiting with a numerical adaptive solver in Section 5, and also give some examples of non-standard size functions involving a varying q.

2. THE GRADIENT LIMITING EQUATION

We now consider how to limit the magnitude of the gradients of a function $h_0(x)$, to obtain a new gradient limited function h(x) satisfying $|\nabla h(x)| \leq g$ everywhere. We require that $h(x) \leq h_0(x)$, and at every x we want h to be as large as possible. We claim that h(x) is the steady-state solution to the following Gradient Limiting Equation:

$$\frac{\partial h}{\partial t} + |\nabla h| = \min(|\nabla h|, g), \tag{1}$$

with initial condition

$$h(t=0, \boldsymbol{x}) = h_0(\boldsymbol{x}). \tag{2}$$

When $|\nabla h| \leq g$, (1) gives that $\partial h/\partial t = 0$, and h will not change with time. When $|\nabla h| > g$, the equation will enforce $|\nabla h| = g$ (locally), and the positive sign multiplying $|\nabla h|$ ensures that information propagates in the direction of increasing values. At steady-state we have that $|\nabla h| = \min(|\nabla h|, g)$, which is the same as $|\nabla h| \leq g$.

For the special case of a convex domain in \mathbb{R}^n and constant g, we can derive an analytical expression for the solution to (1), showing that it is indeed the optimal solution:

Theorem 2.1. Let $\Omega \subset \mathbb{R}^n$ be a bounded convex domain, and I = (0,T) a given time interval. The steady-state solution $h(x) = \lim_{T \to \infty} h(x,T)$ to

$$\begin{cases} \frac{\partial h}{\partial t} + |\nabla h| = \min(|\nabla h|, g) & (x, t) \in \Omega \times I \\ h(x, t)|_{t=0} = h_0(x) & x \in \Omega \end{cases}$$
 (3)

is

$$h(x) = \min_{y} (h_0(y) + g|x - y|). \tag{4}$$

Proof. The Hopf-Lax theorem [12] states that the solution to the Hamilton-Jacobi equation $\frac{du}{dt} + F(\nabla u) = 0$ with initial condition $u(x,0) = u_0(x)$ and convex F(w) is given by

$$u(x,t) = \min_{y} [u_0(y) + tF^*((x-y)/t)],$$
 (5)

where $F^*(u) = \max_w (wu - F(w))$ is the conjugate function of F.

For our equation (3), rewrite as $\frac{\partial h}{\partial t} + F(\nabla h) = 0$, with $F(w) = |w| - \min(|w|, g)$. The conjugate function is

$$F^{*}(u) = \max_{w} (wu - F(w))$$

$$= \max_{w} (wu - |w| + \min(|w|, g))$$

$$= \begin{cases} g|u|, & \text{if } |u| < 1, \\ +\infty & \text{if } |u| \ge 1. \end{cases}$$
(6)

Using (5), we get

$$h(x,t) = \min_{y} \left[h_0(y) + tF^* \left((x-y)/t \right) \right]$$

=
$$\min_{\substack{y \\ |x-y| \le t}} \left(h_0(y) + g|x-y| \right).$$
 (7)

Let $t \to \infty$ to get the steady-state solution to (3):

$$h(x) = \min_{y} (h_0(y) + g|x - y|). \tag{8}$$

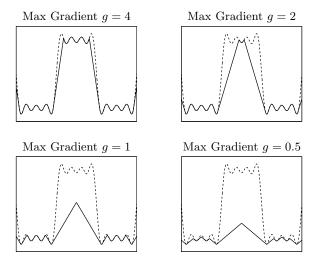


Figure 1: Illustration of gradient limiting by $\partial h/\partial t + |\nabla h| = \min(|\nabla h|, g)$. The dashed lines are the initial conditions h_0 , the solid lines are the gradient limited steady-state solutions h for different parameter values g.

Note that the solution (4) is composed of infinitely many point-source solutions as described before. We could in principle define an algorithm based on (4) for computing h from a given h_0 (both discretized). Such an algorithm would be trivial to implement, but its computational complexity would be proportional to the square of the number of node points. Instead, we solve (3) using efficient Hamilton-Jacobi solvers, see Section 3.

The gradient limiting is illustrated by a 1-D example in Figure 1, where (3) is solved using different values of g and a simple scalar function as initial condition. Note how the large gradients are reduced exactly the amount needed, without affecting regions far away from them. This is very different from traditional smoothing, which affects all data and gives excessive perturbation of the original function $h_0(x)$. Our solution is not necessarily smooth, but it is continuous and $|\nabla h| \leq g$ everywhere.

3. IMPLEMENTATION

One advantage with the continuous formulation of the problem is that a large variety of solvers can be used almost as black-boxes. This includes solvers for structured and unstructured grids, higher-order methods, and specialized fast solvers.

3.1 Cartesian Grids

On a Cartesian background grid, the equation (1) can be solved with just a few lines of code using the following iteration:

$$h_{ijk}^{n+1} = h_{ijk}^n + \Delta t \left(\min(\nabla_{ijk}^+, g) - \nabla_{ijk}^+ \right)$$
 (9)

where

$$\nabla_{ijk}^{+} = \left[\max(D^{-x} h_{ijk}^{n}, 0)^{2} + \min(D^{+x} h_{ijk}^{n}, 0)^{2} + \max(D^{-y} h_{ijk}^{n}, 0)^{2} + \min(D^{+y} h_{ijk}^{n}, 0)^{2} + \max(D^{-z} h_{ijk}^{n}, 0)^{2} + \min(D^{+z} h_{ijk}^{n}, 0)^{2} \right]^{1/2}$$

$$(10)$$

Here, D^{-x} is the backward difference operator in the x-direction, D^{+x} the forward difference operator, etc. The iterations are initialized by $h^0 = h_0$, and we iterate until the updates $\Delta h(x)$ are smaller than a given tolerance. The Δt parameter is chosen to satisfy the CFL-condition, we use $\Delta t = \Delta x/2$. The boundaries of the grid do not need any special treatment since all characteristics point outward.

The iteration (9) converges relatively fast, although the number of iterations grows with the problem size so the total computational complexity is superlinear. Nevertheless, the simplicity makes this a good choice in many situations. If a good initial guess is available, this time-stepping technique might even be superior to other methods. This is the case for problems with moving boundaries, where the size function from the last mesh is likely to be close to the new size function. Another example is numerical adaptivity, when the original size function often has relatively small gradients because of numerical properties of the underlying PDE.

The scheme (9) is first-order accurate in space, and higher accuracy can be achieved by using a second-order solver. See [13] and [14] for details.

3.2 Fast Gradient Limiting

For faster solution of (1) we use a modified version of the fast marching method (Sethian [15], see also Tsitsiklis [16]). The main idea for solving our PDE (1) is based on the fact that the characteristics point in the direction of the gradient, and therefore smaller values are never affected by larger values. This means we can start by fixing the smallest value of the solution, since it will never be modified. We then update the neighbors of this node by a discretization of our PDE, and repeat the procedure. To find the smallest value efficiently we use a min-heap data structure.

During the update, we have to solve for a new h_{ijk} in $\nabla^+_{ijk} = g$, with ∇^+_{ijk} from (10). This expression is simplified by the fact that h_{ijk} should be larger than

Algorithm 1 - Fast Gradient Limiting Description: Solve (1) on a Cartesian grid Input: Initial discretized h_0 , grid spacing Δx Output: Discretized solution hSet $h = h_0$ Insert all h_{ijk} in a min-heap with back pointers while heap not empty Remove smallest element IJK from heap for neighbors ijk of IJK still in heap: compute upwind $|\nabla h_{ijk}|$ if $|\nabla h_{ijk}| > g$ Solve for h_{ijk}^{new} in $\nabla_{ijk}^{+} = g$ from (10) Set $h_{ijk} \leftarrow \min(h_{ijk}, h_{ijk}^{\text{new}})$ end if end for

Table 1: The fast gradient limiting algorithm for Cartesian grids. The computational complexity is $\mathcal{O}(n \log n)$, where n is the number of nodes in the background grid.

end while

all previously fixed values of h, and we can solve a quadratic equation for each octant and set h_{ijk} to the minimum of these solutions.

Our fast algorithm is summarized as pseudo-code in Table 1. Compared to the original fast marching method, we begin by marking all nodes as TRIAL points, and we do not have any FAR points. The actual update involves a nonlinear right-hand side, but it always returns increasing values so the update procedure is valid. The heap is large since all elements are inserted initially, but the access time is still only $\mathcal{O}(\log n)$ for each of the n nodes in the background grid. In total, this gives a solver with computational complexity $\mathcal{O}(n\log n)$. For higher-order accuracy, the technique described in [15] can be applied.

3.3 Unstructured Background Grids

An unstructured background grid gives a more efficient representation of the size function and higher flexibility in terms of node placement. A common choice is to use an initial Delaunay mesh, possibly with a few additional refinements. Several methods have been developed to solve Hamilton-Jacobi equations on unstructured grids, and we have implemented the positive coefficient scheme by Barth and Sethian [17]. The solver is slightly more complicated than the Cartesian variants, but the numerical schemes can essentially be used as black-boxes. A triangulated version of the fast marching method was given in [18], and in [19] the algorithm was generalized to arbitrary node locations.

One particular unstructured background grid is the octree representation, and the Cartesian methods extend naturally to this case (both the iteration and the fast solver). The values are interpolated on the boundaries between cells of different sizes. We mentioned in the introduction that octrees are commonly used to represent size functions, because of the possibility to balance the tree and thereby get a limited variation of cell sizes. Here, we propose to use the octree as a convenient and efficient representation, but the actual values of the size function are computed using our PDE. This gives higher flexibility, for example the possibility to use different values of q.

4. RESULTS

In this section we present several examples of mesh generation using size functions computed with the gradient limiting equation. All triangular and tetrahedral meshes are generated with the smoothing-based mesh generator for distance functions that we presented in [3].

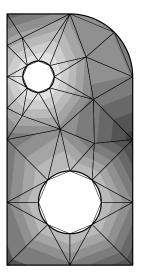
4.1 2-D

We begin with a simple example of gradient limiting in two dimensions on a triangular mesh. For the geometry in Figure 2, we set $h_0(x)$ proportional to the radius of curvature on the boundaries, and to ∞ in the interior. We solve our gradient limiting equation using the positive coefficient scheme to get the mesh size function in the left plot (throughout the paper, light gray corresponds to low values and dark gray to high values). A sample mesh using this result is shown in the right plot.

This example shows that we can apply size constraints in an arbitrary manner, for example only on some of the boundary nodes. The PDE will propagate the values in an optimal way to the remaining nodes, and possibly also change the given values if they violate the grading condition. For this very simple geometry, we can indeed write the size function explicitly as

$$h(x) = \min_{i} (h_i + g\phi_i(\boldsymbol{x})). \tag{11}$$

Here, ϕ_i and h_i are the distance functions and the boundary mesh size for each of the three curved boundaries. But consider, for example, a curved boundary with a non-constant curvature. The analytical expression for the size function of this boundary is non-trivial (it involves the curvature and distance function of the curve). One solution would be to put point-sources at each node of the background mesh, but the complexity of evaluating (11) grows quickly with the number of nodes. By solving our gradient limiting equation, we arrive at the same solution in an efficient and simple way.



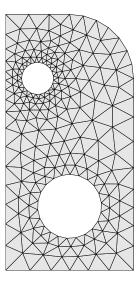


Figure 2: Example of gradient limiting with an unstructured background grid. The size function is given at the curved boundaries and computed by (1) at the remaining nodes.

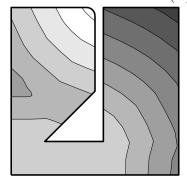
In Figure 3 we show a size function for a geometry with a narrow slit, again generated using the unstructured gradient limiting solver. The initial size function $h_0(x)$ is based on the local feature size and the curved boundary at the top. Note that although the regions on the two sides of the slit are close to each other, the small mesh size at the curved boundary does not influence the other region. This solution is harder to express using source expressions such as (11), and more expensive geometric search routines have to be used instead.

A more complicated example is shown in Figure 4. Here, we have computed the local feature size everywhere in the interior of the geometry. We compute this as the sum of the distance from the boundary and the distance from the medial axis, and we detect the medial axis as the shocks in the distance function (we will present more details about this in a forthcoming paper). The result is stored on a Cartesian grid.

The gradient of the local feature size is greater than g at some places, and we use the fast gradient limiting solver in Algorithm 1 to get a well-behaved size function. We also use curvature adaptation as before. Note that this mesh size function would be very expensive to compute explicitly, since the feature size is defined everywhere in the domain, not just on the boundaries.

As a final example of 2-D mesh generation, we show an object with smooth boundaries in Figure 5. We use a Cartesian grid for the background grid and solve the gradient limiting equation using the fast solver.

Mesh Size Function h(x)



Mesh Based on h(x)

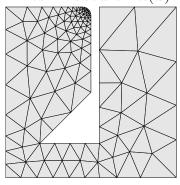
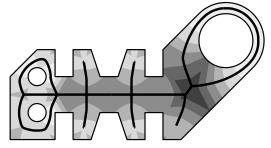


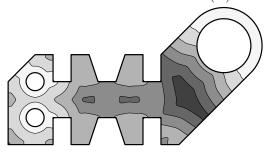
Figure 3: Another example of gradient limiting, showing that non-convex regions are handled correctly. The small sizes at the curved boundary do not affect the region at the right, since there are no connections across the narrow slit.

The feature size is again computed using the medial axis and the distance function, and the curvature is given by the Laplacian of the distance function with a correction since the grid is not aligned with the boundaries

Medial Axis and Feature Size



Mesh Size Function h(x)



Mesh Based on h(x)

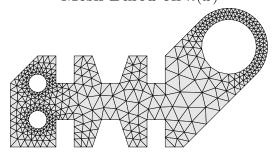
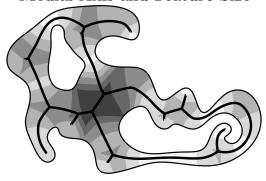
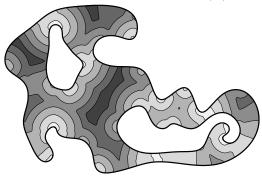


Figure 4: A mesh size function taking into account both feature size, curvature, and gradient limiting. The feature size is computed as the sum of the distance function and the distance to the medial axis.

Medial Axis and Feature Size



Mesh Size Function h(x)



Mesh Based on h(x)

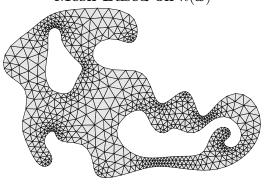
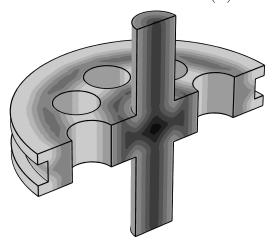


Figure 5: Generation of a mesh size function for a geometry with smooth boundaries.

Mesh Size Function h(x)



Mesh Based on h(x)

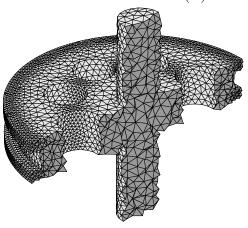


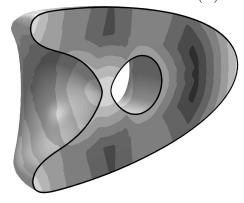
Figure 6: Cross-sections of a 3-D mesh size function and a sample tetrahedral mesh.

4.2 3-D

The PDE-based formulation generalizes to arbitrary dimensions, and in Figure 6 we show a 3-D example. Here, the feature size is computed explicitly from the geometry description, the curvature adaptation is applied on the boundary nodes, and the size function is computed by gradient limiting with g=0.2. This results in a well-shaped tetrahedral mesh, in the bottom plot (slivers have been removed by face swapping and edge flipping).

The smooth surface of the geometry in Figure 7 is represented by its discretized signed distance function. We apply gradient limiting with g=0.25 on a size function that is computed automatically, tak-

Mesh Size Function h(x)



Mesh Based on h(x)

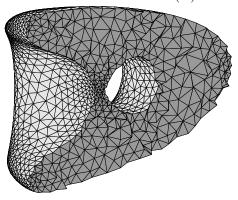


Figure 7: Cross-sections of a 3-D mesh size function and a sample tetrahedral mesh. Note the small elements in the narrow region, given by the local feature size, and the smooth increase in element sizes.

ing into account curvature adaptation and feature size adaptation (from the medial axis, as described before). Again, the plots show cross-sections of the final mesh size function and an example mesh.

4.3 Performance and Accuracy

To study the performance and the accuracy of our algorithms, we consider a simple model problem in $\Omega = (-50, 50) \times (-50, 50)$ with two point-sources, h(-10,0) = 1 and h(10,0) = 5, and g = 0.3. The true solution is given by (4), and we solve the problem on a Cartesian grid of varying resolution.

In Table 2 we compare the execution times for three different solvers – edge-based iterations, Hamilton-Jacobi iterations, and the Hamilton-Jacobi fast gradient limiting solver. The edge-based iterative solver loops until convergence over all neighboring nodes

# Nodes	Edge Iter.	H-J Iter.	H-J Fast
10,000	0.009s	0.060 s	0.006s
40,000	0.068s	0.470 s	0.030s
160,000	0.844s	3.625s	0.181s
640,000	$6.609 \mathrm{s}$	28.422s	1.453s

Table 2: Performance of the edge-based iterative solver, the Hamilton-Jacobi iterative solver, and the Hamilton-Jacobi fast gradient limiting solver.

i, j and updates the size function locally by $h_j \leftarrow \min(h_j, h_i + g|\mathbf{x}_j - \mathbf{x}_i|)$ (assuming $h_j > h_i$). The iterative Hamilton-Jacobi solver is based on the iteration (9) with a tolerance of about two digits. All algorithms are implemented in C++ using the same optimizations, and the tests were done on a PC with an Athlon XP 2800+ processor.

The table shows that the iterative Hamilton-Jacobi solver is about five times slower than the simple edge-based iterations. This is because the update formula for the edge-based iterations is simpler (all edge lengths are the same) and since the Hamilton-Jacobi solver requires more iterations for high accuracy (although their asymptotic behavior should be the same). The fast solver is better than the iterative solvers, and the difference gets bigger with increasing problem size (since it is asymptotically faster). Note that these background meshes are relatively large and that all solvers probably are sufficiently fast in many practical situations.

We also mention that simple algorithms based on the explicit expression (4) for convex domains or geometric searches for non-convex domains might be faster for a small number of point-sources. However, these methods are not practical for larger problems because of the $\mathcal{O}(n^2)$ complexity.

Next we compare the accuracy of the edge-based solver and Hamilton-Jacobi discretizations of first and second order accuracy. The true solution is given by (4), and an algorithm based on this expression would of course be exact to full precision. Figure 8 shows solutions for a 100×100 grid, and it is clear that the edge-based solver is highly inaccurate since it does not take into account the continuous nature of the problem. It has a maximum error of 7.79, compared to 0.38 and 0.10 for the Hamilton-Jacobi solvers. This is similar to the error in solving the Eikonal equation using Dijkstra's shortest path algorithm instead of the continuous fast marching method [15]. The error with the edge-based solver might be even larger for unstructured background meshes which often have low element qualities.







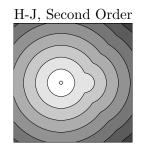


Figure 8: Comparison of the accuracy of the discrete edge-based solver and the continuous Hamilton-Jacobi solver on a Cartesian background mesh. The edge-based solver does not capture the continuous nature of the propagating fronts.

5. OTHER APPLICATIONS

In this section we show two special applications of the gradient limiting equation – numerical adaptation and non-constant g values.

5.1 Numerical Adaptation

Numerical adaptation is a technique for solving PDEs using mesh size functions that are automatically generated to reduce the discretization error. From an error estimator in each element, a new mesh size function is computed. The mesh can then be updated, either by local refinements or remeshing. The procedure is repeated until the desired accuracy is achieved.

One problem when regenerating the mesh is that the size function h(x) from the adaptive solver might be highly irregular. The error estimation often varies between neighboring elements, giving high gradients also in the size function. A simple solution is to smooth the size function, e.g. using Laplacian smoothing. However, this introduces large deviations from the original size function, even where the gradient is small. A better method is to use gradient limiting and solve (1) on the same unstructured mesh that the size function is defined on.

An example is shown in Figure 9. We define a velocity

$$\mathbf{v} = [1, -2\pi A \cos 2\pi x] \tag{12}$$

(with A=0.3) and solve for u in the following advection problem:

$$\mathbf{v} \cdot \nabla u(x, y) = 0, \quad (x, y) \in (-1, 1) \times (-1, 1) \quad (13)$$

with boundary conditions

$$u(x, -1) = 0, (14)$$

$$u(x,1) = 1, (15)$$

$$u(-1, y) = \begin{cases} 1, & \text{if } y \ge 0\\ 0, & \text{if } y < 0. \end{cases}$$
 (16)

The exact solution to this problem has a jump:

$$u(x,y) = \begin{cases} 1, & \text{if } y \ge A \sin 2\pi x \\ 0, & \text{if } y < A \sin 2\pi x. \end{cases}$$
 (17)

We discretize (13)-(16) using piecewise linear finite elements with streamline-diffusion stabilization. To obtain an accurate numerical solution, the discontinuity along $y = A \sin 2\pi x$ has to be resolved. We do this using numerical adaptation in the L_2 -norm, see [20]. The size function from the adaptive scheme is highly irregular, and in particular it specifies large variations in element sizes which would give low-quality triangles. After gradient limiting the mesh size function is well-behaved and a high-quality mesh can be generated (in the bottom plot of Figure 9).

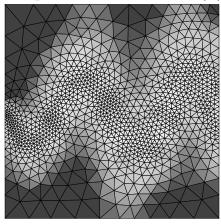
5.2 Space and Solution Dependent g

The solution of the gradient limiting equation remains well-defined if we make g(x) a function of space. The numerical schemes in Section 3 are still valid, and we replace for example g in (9) with g_{ijk} . An application of this is when some regions of the geometry require higher element qualities, and therefore also a smaller maximum gradient in the size function.

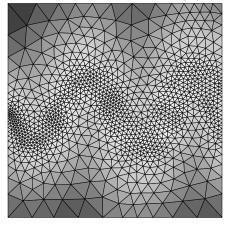
Figure 10 shows a simple example. The initial mesh size h_0 is based on curvatures and feature sizes. The left and the right parts of the region have different values of g, and the gradient limiting generates a new size function h satisfying $|\nabla h| \leq g(x)$ everywhere.

Another possible extension is to let g be a function of the solution h(x) (although it is then not clear if the gradient limiting equation has one unique solution). This can be used, for example, to get a fast increase for small element sizes but smaller variations for large elements. In a numerical solver this might be compensated by the smaller truncation error for the small elements. A simple example is shown in Figure 11, where g(h) varies smoothly between 0.6 (for small elements) and 0.2 (for large elements).

Adaptive Size Function h(x)



Gradient Limited h(x)



New Mesh and Solution u(x)

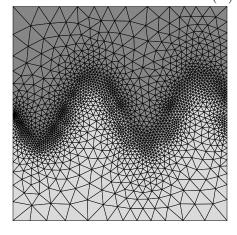
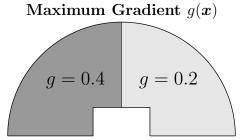
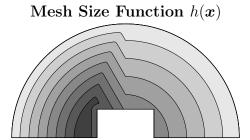


Figure 9: An example of numerical adaptation for solution of (13)-(16).





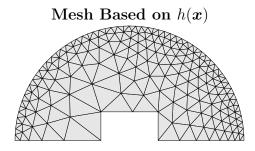


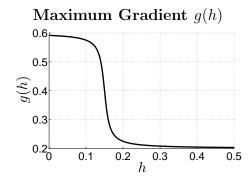
Figure 10: Gradient limiting with space-dependent g(x).

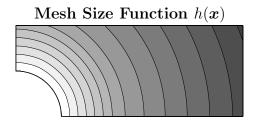
In the iterative solver, we replace g with $g(h_{ijk})$, and if the iterations converge we have obtained a solution. In the fast solver, we solve a (scalar) non-linear equation $\nabla_{ijk}^+ = g(h_{ijk})$ at every update.

6. CONCLUSIONS

We have presented a new, continuous formulation of the gradient limiting procedure, which is an important part in the generation of good mesh size functions. For convex domains, we showed that the continuous solution is the optimal minimum over infinitely many point-sources. The discretized equation can be solved efficiently on all types of background meshes in any dimension. We showed several examples of high-quality meshes generated with our mesh size functions, and we gave an example of gradient limiting for an adaptive finite element solver.

We give several suggestions for future development:





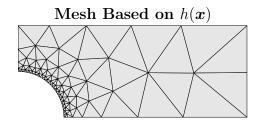


Figure 11: Gradient limiting with solution-dependent g(h). The distances between the level sets of h(x) are smaller for small h, giving a faster increase in mesh size.

- Implementing a fast marching based solver for triangular/tetrahedral background meshes. The methods described in [18] and [19] should be applicable in a straightforward way.
- Extending the method to anisotropic mesh size functions. There might be a PDE similar to the gradient limiting equation (or a system of PDEs) based on general metrics [9].
- Adaptive generation of background meshes. Zhu et al [7] discussed an intuitive, iterative approach for refinement of background meshes. With our PDE-based formulation, we can achieve this in a strict and systematic way by applying error estimators for numerical adaptive solvers [20] on the discretized solution h(x).

References

- Peraire J., Vahdati M., Morgan K., Zienkiewicz O.C. "Adaptive Remeshing for Compressible Flow Computations." *Journal of Computational Physics*, vol. 72, no. 2, 449–466, 1987
- [2] Blacker T.D., Stephenson M.B. "Paving: A New Approach to Automated Quadrilateral Mesh Generation." International Journal For Numerical Methods in Engineering, vol. 32, 811–847, 1991
- [3] Persson P.O., Strang G. "A Simple Mesh Generator in MATLAB." SIAM Review, vol. 46, no. 2, 329–345, June 2004
- [4] Ruppert J. "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation." *Journal of Algorithms*, vol. 18, no. 3, 548–585, 1995
- [5] Shewchuk J.R. "Delaunay Refinement Algorithms for Triangular Mesh Generation." Computational Geometry: Theory and Applications, vol. 22, no. 1-3, 21–74, May 2002
- [6] Owen S., Saigal S. "Surface Mesh Sizing Control." International Journal For Numerical Methods in Engineering, vol. 47, 497–511, 2000
- [7] Zhu J., Blacker T., Smith R. "Background Overlay Grid Size Functions." Proceedings of 11th International Meshing Roundtable, pp. 65–74. Sandia Nat. Lab., September 2002
- [8] Zhu J. "A New Type of Size Function Respecting Premeshed Entities." Proceedings of 11th International Meshing Roundtable, pp. 403–413. Sandia Nat. Lab., September 2003
- [9] Borouchaki H., Hecht F., Frey P.J. "Mesh Gradation Control." Proceedings of 6th International Meshing Roundtable, pp. 131–141. Sandia Nat. Lab., October 1997
- [10] Frey P.J., Marechal L. "Fast Adaptive Quadtree Mesh Generation." Proceedings of 7th International Meshing Roundtable, pp. 211–224. Sandia Nat. Lab., October 1998
- [11] Yerry M.A., Shephard M.S. "A Modified Quadtree Approach to Finite Element Mesh Generation." *IEEE Computer Graphics and Applica*tions, vol. 3, no. 1, 39–46, 1983
- [12] Hopf E. "Generalized Solutions of Non-Linear Equations of First Order." Journal of Mathematics and Mechanics, vol. 14, 951–973, 1965

- [13] Osher S., Sethian J.A. "Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations." *Journal of Computational Physics*, vol. 79, no. 1, 12–49, 1988
- [14] Harten A., Engquist B., Osher S., Chakravarthy S.R. "Uniformly High Order Accurate Essentially Non-Oscillatory Schemes." *Journal of Computa*tional Physics, vol. 71, no. 2, 231–303, 1987
- [15] Sethian J. "A Fast Marching Level Set Method for Monotonically Advancing Fronts." Proceedings of the National Academy of Sciences, vol. 93 (4), pp. 1591–1595. 1996
- [16] Tsitsiklis J.N. "Efficient Algorithms for Globally Optimal Trajectories." *IEEE Transactions on Automatic Control*, vol. 40, 1528–1538, September 1995
- [17] Barth T.J., Sethian J.A. "Numerical Schemes for the Hamilton-Jacobi and Level Set Equations on Triangulated Domains." *Journal of Computa*tional Physics, vol. 145, 1–40, 1998
- [18] Kimmel R., Sethian J. "Fast Marching Methods on Triangulated Domains." Proceedings of the National Academy of Sciences, vol. 95, pp. 8341–8435. 1998
- [19] Covello P., Rodrigue G. "A Generalized Front Marching Algorithm for the Solution of the Eikonal Equation." Journal of Computational Physics, vol. 156, no. 2, 371–388, 2003
- [20] Eriksson K., Estep D., Hansbo P., Johnson C. Computational Differential Equations. Press Syndicate of the University of Cambridge, 1996